Übungen zum Kapitel 10

Aggregation von Daten und Gruppenoperationen

Erstellt und überarbeitet: armin.baenziger@zhaw.ch, 5. März 2020

In [1]:
```
%autosave 0
```

```
Autosave disabled
```

(A.1) Laden Sie NumPy, Pandas und Matplotlib.pyplot mit der üblichen Abkürzung.

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

(A.2) Führen Sie den Magic Command aus, so dass Matplotlib-Plots "inline" erscheinen.

In [3]:
```
%matplotlib inline
```

Im Ordner "weitere_Daten" finden Sie den Datensatz `Auto.csv` den wir bereits im letzten Kapitel verwenden haben. Die folgende Zelle lädt die Daten ins DataFrame `Auto` , setzt den Autonamen (`name`) als Index und codiert die Variable `origin` um.

In [4]:
```
Auto = pd.read_csv('../weitere_Daten/Auto.csv',
                   sep=';', index_col=8)
Auto.origin.replace({1: 'USA', 2: 'Europe', 3: 'Japan'},
                   inplace=True)
```

(B.1) Selektieren Sie die *letzten acht* Zeilen des DataFrames für die Bildschirmausgabe.

In [5]:
```
Auto.tail(8)
```

Out[5]:

name	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin
toyota celica gt	32.0	4	144.0	96	2665	13.9	82	Japan
dodge charger 2.2	36.0	4	135.0	84	2370	13.0	82	USA
chevrolet camaro	27.0	4	151.0	90	2950	17.3	82	USA
ford mustang gl	27.0	4	140.0	86	2790	15.6	82	USA
vw pickup	44.0	4	97.0	52	2130	24.6	82	Europe
dodge rampage	32.0	4	135.0	84	2295	11.6	82	USA
ford ranger	28.0	4	120.0	79	2625	18.6	82	USA
chevy s-10	31.0	4	119.0	82	2720	19.4	82	USA

(B.2) Die Methode `info` gibt kurze Informationen zu einem DataFrame, einschliesslich des Index- und des Spalten-Datentyps (*dtypes*), der Anzahl Nicht-Nullwerte und der Speicherauslastung. Führen Sie nun `Auto.info()` aus.

In [6]:
```
Auto.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 392 entries, chevrolet chevelle malibu to chevy s-10
Data columns (total 8 columns):
mpg             392 non-null float64
cylinders       392 non-null int64
displacement    392 non-null float64
horsepower      392 non-null int64
weight          392 non-null int64
acceleration    392 non-null float64
year            392 non-null int64
origin          392 non-null object
dtypes: float64(3), int64(4), object(1)
memory usage: 27.6+ KB
```

(B.3) Berechnen Sie die mittlere Anzahl Zylinder (`cylinders`) nach Herkunftsgebiet (`origin`) mittels der Methode `groupby` . Was stellen Sie fest?

In [7]:
```
Auto.cylinders.groupby(Auto.origin).mean()
```

Out[7]:
```
origin
Europe    4.161765
Japan     4.101266
USA       6.277551
Name: cylinders, dtype: float64
```

In den untersuchten Jahren (bzw. den Autos des Datensatzes) wurden in US-Autos durchschnittlich etwa 2 Zylindern mehr verbaut als in Autos aus Europa oder Japan.

(B.4) Berechnen Sie die mittleren Meilen pro Galone (`mpg`) nach Baujahr der Autos (`year`) und der Herkunftsregion (`origin`). Speichern Sie das Ergebnis im Objekt `Entwicklung_MPG` .

In [8]:
```
Entwicklung_MPG = Auto.mpg.groupby(
    [Auto.year, Auto.origin]).mean()
Entwicklung_MPG.head(9)
```

Out[8]:
```
year  origin
70    Europe    25.200000
      Japan     25.500000
      USA       15.272727
71    Europe    28.750000
      Japan     29.500000
      USA       17.736842
72    Europe    22.000000
      Japan     24.200000
      USA       16.277778
Name: mpg, dtype: float64
```

(B.5) Verwenden Sie die Methode `unstack` auf die Series `Entwicklung_MPG`, welche Sie gerade erstellt haben.

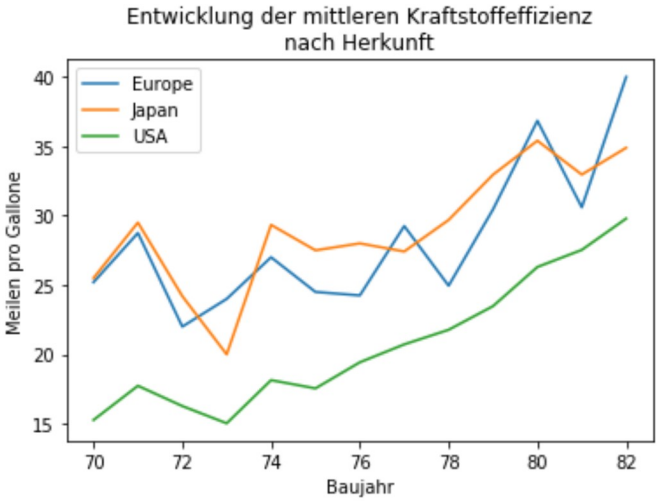
In [9]:
```
Entwicklung_MPG.unstack()
```

Out[9]:

origin	Europe	Japan	USA
year			
70	25.2000	25.500000	15.272727
71	28.7500	29.500000	17.736842
72	22.0000	24.200000	16.277778
73	24.0000	20.000000	15.034483
74	27.0000	29.333333	18.142857
75	24.5000	27.500000	17.550000
76	24.2500	28.000000	19.431818
77	29.2500	27.416667	20.722222
78	24.9500	29.687500	21.772727
79	30.4500	32.950000	23.478261
80	36.8375	35.400000	26.300000
81	30.6000	32.958333	27.530769
82	40.0000	34.888889	29.789474

(B.6) Stellen Sie die gerade erstellte Entwicklung der Kraftstoffeffizienz (nach Baujahr), separat nach Herkunftsregion, in *einem* Liniendiagramm dar. Was stellen Sie fest?

In [10]:
```
Entwicklung_MPG.unstack().plot(title=
        'Entwicklung der mittleren Kraftstoffeffizienz\nnach Herkunft')
# Weitere Einstellungen möglich:
plt.xlabel('Baujahr')
plt.ylabel('Meilen pro Gallone')
plt.legend();
```



Die Kraftsoffeffizienz der Autos hat etwa ab der ersten Erdölkrise (1973) in allen drei Herstellerländern stark zugenommen, wobei US-Autos etwa 8 Meilen weniger weit mit einer Gallone Kraftstoff fahren als Autos aus Europa oder Japan.

(C.1) Laden Sie die *simulierten* Lohndaten `dflohn.pkl` im Ordner "weitere_Daten" in das DataFrame `dflohn`.

In [11]:
```
dflohn = pd.read_pickle('../weitere_Daten/dflohn.pkl')
dflohn.head()
```

Out[11]:

Person	Lohn	Geschlecht	Alter	Zivilstand
1	4107.0	m	40	g
2	5454.0	m	47	vw
3	3719.0	m	41	g
4	6194.0	m	18	v
5	NaN	m	27	v

(C.2) Gruppieren Sie das DataFrame `dflohn` nach Geschlecht und Zivilstand. Berechnen Sie dann für jede Gruppe den Medianlohn. Setzen Sie hierzu die `groupby` -Methode ein.

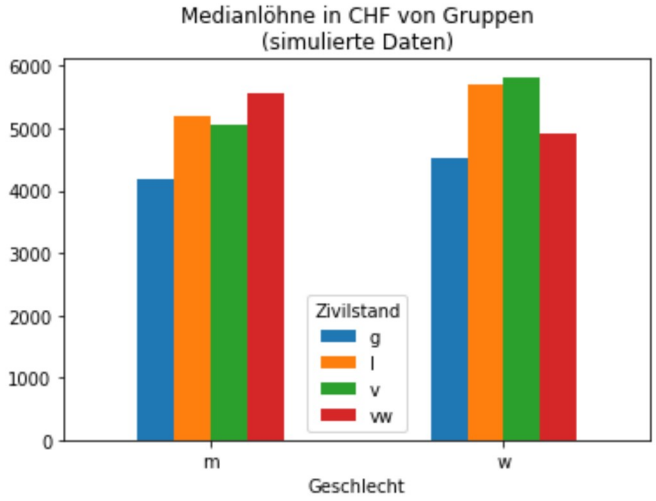
In [12]:
```
gruppiert = dflohn.Lohn.groupby(
                [dflohn.Geschlecht,
                 dflohn.Zivilstand])
gruppiert.median()
```

Out[12]:
```
Geschlecht  Zivilstand
m           g             4173.0
            l             5210.5
            v             5058.0
            vw            5555.5
w           g             4515.5
            l             5701.0
            v             5824.0
            vw            4920.5
Name: Lohn, dtype: float64
```

(C.3) Stellen Sie die soeben berechneten Medianlöhne nach Gruppenzugehörigkeit in einem gruppierten Säulendiagramm dar. Tipp: Sie müssen zuerst die `unstack` -Methode auf die Resultate-Series (mit dem hierarchischen Index) anwenden.

In [13]:
```
gruppiert.median().unstack().plot.bar(
    title='Medianlöhne in CHF von Gruppen' +
          '\n(simulierte Daten)',
    rot=0)
```

Out[13]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x275dc008588>
```



(C.4) Berechnen Sie je für Frauen und Männer in `dflohn` das erste, zweite und dritte Quartil (bzw. 25%-, 50%, 75%-Quantil) der Löhne.

In [14]:
```
resultat = dflohn.Lohn.groupby(
           dflohn.Geschlecht).quantile(
           [0.25, 0.5, 0.75])
resultat.index.names = ('Geschlecht', 'Quantile')
resultat.unstack()
# Die letzten zwei Zeilen sind "Kosmetik".
```

Out[14]:

Quantile	0.25	0.5	0.75
Geschlecht			
m	4356.0	5454.0	6488.0
w	4041.0	5053.0	7174.5

(C.5) *Exkurs:* Berechnen Sie je für Frauen und Männer in `dflohn` den Mittelwert und den Median der Löhne und beschriften Sie diese Statistiken in der Tabelle entsprechend. Tipp: Methode `agg` verwenden.

In [15]:
```
dflohn.Lohn.groupby(dflohn.Geschlecht).agg(
    ['mean', 'median'])
```

Out[15]:

Geschlecht	mean	median
m	5838.918367	5454.0
w	5850.380000	5053.0

In [16]:
```
# Hie noch eine Lösung mit angepassten
# Spaltenüberschriften:
dflohn.Lohn.groupby(dflohn.Geschlecht).agg(
    [('Mittelwert', 'mean'), ('Median', 'median')])
```

Out[16]:

Geschlecht	Mittelwert	Median
m	5838.918367	5454.0
w	5850.380000	5053.0

In [17]:
```
# Noch ein Lösungsvorschlag:
dflohn.Lohn.groupby(dflohn.Geschlecht).describe()
# Der Median ist das 50%-Quantil im Output.
```

Out[17]:

Geschlecht	count	mean	std	min	25%	50%	75%	max
m	49.0	5838.918367	2547.627154	1894.0	4356.0	5454.0	6488.0	16502.0
w	50.0	5850.380000	2671.756931	2106.0	4041.0	5053.0	7174.5	15911.0

(D.1) Die folgende Zeile lädt Index-Daten von SIX Swiss Exchange (https://www.six-swiss-exchange.com/indices/data_centre/shares/spi_de.html (https://www.six-swiss-exchange.com/indices/data_centre/shares/spi_de.html)). Stellen Sie sicher, dass Sie die Bedeutung von allen Funktions-Argumenten verstehen.

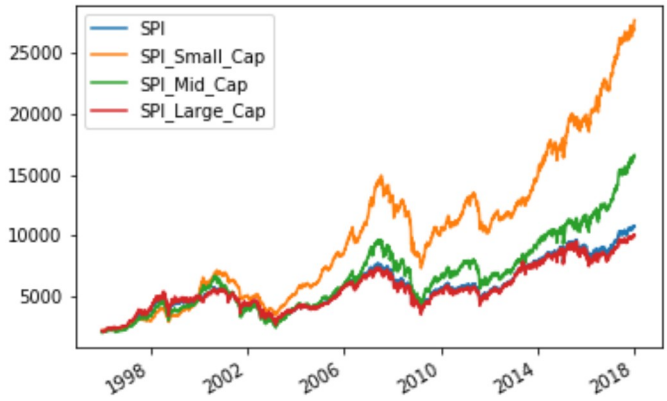
In [18]:
```
Indizes = pd.read_csv('../weitere_Daten/hspitr.csv',
                      sep=';', index_col=0, parse_dates=True)
# Mit parse_daes=True wird versucht, den Index
# als Datum (nicht String) umzuwandeln.
Indizes.sort_index(inplace=True)
Indizes.head()
```

Out[18]:

Date	SPI	SPI_Small_Cap	SPI_Mid_Cap	SPI_Large_Cap
1996-01-03	2172.31	2170.14	2040.95	2208.62
1996-01-04	2182.67	2184.72	2049.35	2219.26
1996-01-05	2175.09	2181.00	2054.88	2208.69
1996-01-08	2176.68	2194.96	2071.98	2206.46
1996-01-09	2174.13	2204.00	2077.84	2201.63

(D.2) Stellen Sie den Verlauf der vier Zeitreihen in einem Diagramm dar.

In [19]:
```
Indizes.plot()
plt.xlabel('')
plt.show()
```



Man erkennt sehr deutlich (trotz geringfügig unterschiedlichem Startwert), dass sich der Teilindex der kleinkapitalisierten Unternehmungen viel besser entwickelt hat als jener der mittelgrossen Unternehmungen und dieser wiederum besser als der Teilindex der grosskapitalisierten Unternehmungen.

Der Index der grosskapitalisierten Unternehmungen entwickelte sich fast identisch zum Gesamtindex (SPI), was darauf zurückzuführen ist, dass die Gewichtung der Large-Caps im Gesamtindex sehr hoch ist.

(D.3) Erstellen Sie das DataFrame `Renditen`, welches die (diskreten) Tagesrenditen der vier Indexreihen umfasst. Verwenden Sie hierzu die Methode `pct_change`. Beseitigen Sie (die entstandenen) Fehlwerte mit der Methode `dropna`.

In [20]:
```
Renditen = Indizes.pct_change().dropna()
Renditen.head()
```

Out[20]:

Date	SPI	SPI_Small_Cap	SPI_Mid_Cap	SPI_Large_Cap
1996-01-04	0.004769	0.006718	0.004116	0.004817
1996-01-05	-0.003473	-0.001703	0.002698	-0.004763
1996-01-08	0.000731	0.006401	0.008322	-0.001010
1996-01-09	-0.001172	0.004119	0.002828	-0.002189
1996-01-10	-0.011490	-0.011148	-0.004380	-0.012945

(D.4) Charakterisieren Sie die Renditeverteilungen durch den Mittelwert und die Standardabweichung der Tagesrenditen. Was stellen Sie fest? Tipp: Methode `describe` verwenden.

In [21]:
```
Renditen.describe()[1:3]
```

Out[21]:

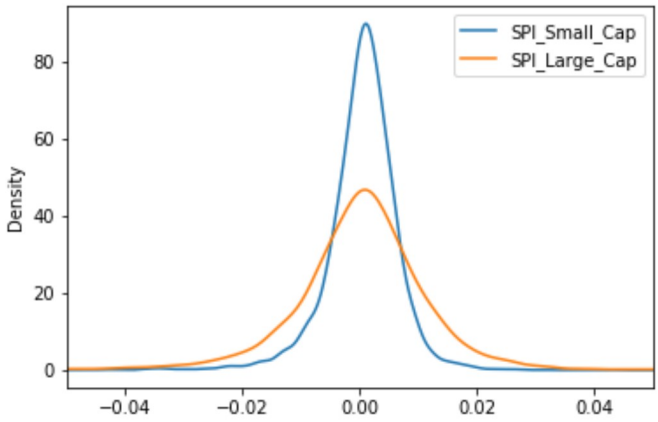
	SPI	SPI_Small_Cap	SPI_Mid_Cap	SPI_Large_Cap
mean	0.000351	0.000479	0.000424	0.000343
std	0.011159	0.006058	0.009555	0.011836

Obwohl die mittlere Rendite bei den kleinkapitalisierten Unternehmungen grösser ist als bei den grosskapitalisierten, scheint das Risiko, gemessen an der Standardabweichung (Volatilität) *kleiner* zu sein bei kleinkapitalisierten Werten. (Es wäre aber zu untersuchen, inwiefern geringer Handel bei den kleinkapitalisierten Werten zu *scheinbar* wenig volatilen Renditen führte.)

(D.5) Stellen Sie die zwei Verteilungen der Tagresrenditen des SPI-Large-Cap und des SPI-Small-Cap in einem Diagramm dar. Wählen Sie hierzu Kernel-Density-Plots (`.plot.density()`) und einen Renditebereich von -5% bis 5% (`xlim=[-0.05, 0.05]`).

In [22]:
```
Renditen[['SPI_Small_Cap', 'SPI_Large_Cap']
        ].plot.density(xlim=[-0.05,0.05])
```

Out[22]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x275dc1b2198>
```



Man sieht deutlich die geringere Streuung der Small-Cap-Renditen im Vergleich zu den Large-Cap-Renditen.

(D.6) Mit `Renditen.index.year` können die einzelnen *Jahre* aus dem Datum (Index) extrahiert werden. (Mehr dazu erfahren Sie im nächsten Kapitel.) Berechnen Sie hiermit die Median-Renditen der vier Indizes *gruppiert nach Jahren*.

In [23]:
```
Renditen.groupby(Renditen.index.year).median().head()
```

Out[23]:

Date	SPI	SPI_Small_Cap	SPI_Mid_Cap	SPI_Large_Cap
1996	0.001243	0.000587	0.000656	0.001304
1997	0.002545	0.001242	0.002146	0.002639
1998	0.001506	0.001565	0.001899	0.001126
1999	0.000645	0.001771	0.001101	0.000493
2000	0.000511	0.000395	0.001975	0.000143

(D.7) Berechnen Sie die Standardabweichungen der Renditen der Indizes *gruppiert nach Jahren*.

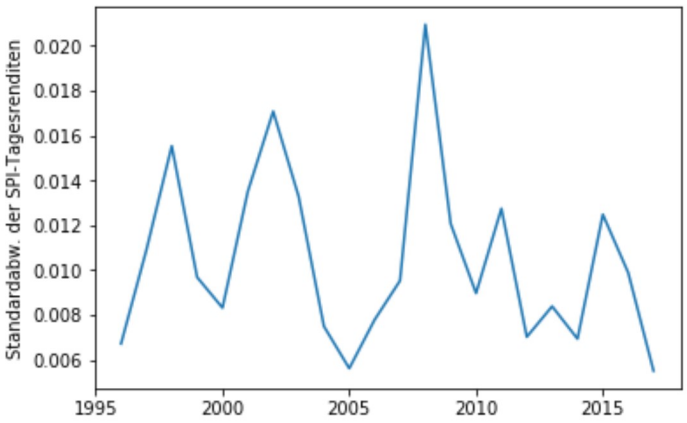
In [24]:
```
std_jahr = Renditen.groupby(Renditen.index.year).std()
std_jahr.head()
```

Out[24]:

Date	SPI	SPI_Small_Cap	SPI_Mid_Cap	SPI_Large_Cap
1996	0.006731	0.003726	0.004492	0.007625
1997	0.010907	0.005753	0.007899	0.011952
1998	0.015532	0.007678	0.011252	0.016670
1999	0.009689	0.004986	0.005327	0.010651
2000	0.008327	0.008390	0.009566	0.009238

(D.8) Stellen Sie die Entwicklung der Standardabweichungen des SPI pro Jahr in einem Liniendiagramm dar. Was stellen Sie fest?

In [25]:
```
std_jahr.SPI.plot();
plt.xlabel('')
plt.ylabel('Standardabw. der SPI-Tagesrenditen');
```



In Krisenzeiten stieg die Volatilität stark an, insb. während der globalen Finanzkrise ab 2008.

Zum Schluss untersuchen wir nochmals den Datensatz `Auto` .

In [26]:
```
# Ausgangslage:
Auto.head()
```

Out[26]:

name	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin
chevrolet chevelle malibu	18.0	8	307.0	130	3504	12.0	70	USA
buick skylark 320	15.0	8	350.0	165	3693	11.5	70	USA
plymouth satellite	18.0	8	318.0	150	3436	11.0	70	USA
amc rebel sst	16.0	8	304.0	150	3433	12.0	70	USA
ford torino	17.0	8	302.0	140	3449	10.5	70	USA

(E.1) Erstellen Sie eine (eindimensionale) Häufigkeitsverteilung der kategorialen Variable `origin` (Herstellerland).

In [27]:
```
Auto.origin.value_counts()
```

Out[27]:
```
USA       245
Japan      79
Europe     68
Name: origin, dtype: int64
```

Die meisten Autos im Datensatz stammen aus den USA.

Wir erstellen nun eine (zweidimensionale) Kreuztabelle, welche die empirische Verteilung bezüglich Herstellerland (`origin`) und Anzahl Zylinder (`cylinders`) aufzeigt.

In [28]:
```
pd.crosstab(Auto.origin, Auto.cylinders, margins=True)
```

Out[28]:

origin	cylinders 3	4	5	6	8	All
Europe	0	61	3	4	0	68
Japan	4	69	0	6	0	79
USA	0	69	0	73	103	245
All	4	199	3	83	103	392

Lesebeispiel: Es gibt 61 Autos, die aus Europa kommen *und* 4 Zylinder haben.

(E.2) Erstellen Sie die gleiche Tabelle wie oben aber mit *relativen* statt aboluten Häufigkeiten. Runden Sie die relativen Häufigkeiten auf zwei Nachkommastellen.

In [29]:
```
pd.crosstab(Auto.origin, Auto.cylinders,
            margins=True, normalize=True).round(2)
```

Out[29]:

cylinders	3	4	5	6	8	All
origin						
Europe	0.00	0.16	0.01	0.01	0.00	0.17
Japan	0.01	0.18	0.00	0.02	0.00	0.20
USA	0.00	0.18	0.00	0.19	0.26	0.62
All	0.01	0.51	0.01	0.21	0.26	1.00

Lesebeispiel: 16 Prozent der Autos (im Datensatz) stammen aus Europa und haben 4 Zylinder.

(E.3) Wie sind die relativen Häufigkeiten der folgenden Tabelle zu interpretieren?

In [30]:
```
tab = pd.crosstab(Auto.origin, Auto.cylinders,
            normalize='columns', margins=True)
tab.round(2)
```

Out[30]:

cylinders	3	4	5	6	8	All
origin						
Europe	0.0	0.31	1.0	0.05	0.0	0.17
Japan	1.0	0.35	0.0	0.07	0.0	0.20
USA	0.0	0.35	0.0	0.88	1.0	0.62

Es handelt sich um eine bedingte Häufigkeitsverteilung (bedingt auf die Anzahl Zylinder). Beispielsweise sind 100 Prozent der Autos mit drei Zylindern aus Japan. Bei den Autos mit 4 Zylindern stammt etwa je ein Drittel aus Europa, Japan und den USA. 5 Zylinder gab es nur in europäischen Autos (im Datensatz); 6 oder 8 Zylinder gab es fast nur bzw. nur in US-Autos (im Datensatz).

(E.4) Berechnen Sie nun nochmals die mittleren Meilen pro Galone (`mpg`) nach Baujahr der Autos (`year`) und der Herkunftsregion (`origin`). Verwenden Sie nun die Methode `pivot_table` statt `groupby` .

In [31]:
```
Auto.pivot_table(values  = 'mpg',
                 index   = 'year',
                 columns = 'origin').round(2)
```

Out[31]:

origin	Europe	Japan	USA
year			
70	25.20	25.50	15.27
71	28.75	29.50	17.74
72	22.00	24.20	16.28
73	24.00	20.00	15.03
74	27.00	29.33	18.14
75	24.50	27.50	17.55
76	24.25	28.00	19.43
77	29.25	27.42	20.72
78	24.95	29.69	21.77
79	30.45	32.95	23.48
80	36.84	35.40	26.30
81	30.60	32.96	27.53
82	40.00	34.89	29.79

Ende der Übung